MESSAGE HANDLING

Reference to Related Applications

This application claims priority to co-pending U.S. Provisional Application Serial No. 60/257,313, filed December 20, 2000, entitled "Message Handling".

Background

Applications running on different network computers often share information. For example, an application running at one computer may feed stock market data to applications at remote computers. To share information, applications often use a messaging system. A messaging system handles a wide variety of tasks associated with application and network communication. These systems often ease application development by shielding programmers from the details of message handling.

One type of messaging system is known as "message-oriented middleware." Essentially, message-oriented middleware systems receive messages from one application and store these messages until retrieved by another application. This scheme can free applications to perform other operations until they choose to receive a message.

Some messages do not require reliable transmission. For example, the loss of a message containing a stock quote may not be of particular concern if other messages with the quote quickly follow. More critical data exchanges, however, may require greater reliability. Thus, many messaging systems support "guaranteed messages" that the messaging system promises not to lose. Typically, after receiving a guaranteed message, message-oriented middleware systems transmit an acknowledgment message to the sending application indicating that from then on,

the system will not lose the message. To provide this guarantee, many messaging systems save guaranteed messages in persistent storage such as a hard disk or other non-volatile storage medium. Thus, even in the event of a messaging system problem, such as a system crash, the system can still access and deliver the guaranteed messages when the system resumes operation.

Summary

In general, in one aspect, the disclosure describes a method of handling messages received at a messaging system server. The method includes storing, in non-persistent storage, messages received from at least one client, removing delivered messages from the non-persistent storage, and saving messages stored in the non-persistent storage to persistent storage after a delay interval.

Embodiments many include one or more of the following features. The storing, removing, and saving may occur concurrently. Storing in non-persistent storage may include storing in a log queue. The messages may be guaranteed messages. The messaging system may be a message-oriented middleware system.

The method may further include transmitting an acknowledgement message that indicates that the received message will not be lost by the server in the case of server failure. Transmitting may include transmitting the acknowledgment message to the client for a delivered message or storage of the message in persistent storage.

The method may further include determining the delay interval. For example, such determining may occur by determining at least one metric based on messages handled by the server and determining the delay interval based on the at least

one metric. The metric may include a metric based on a number of sending clients using the server to deliver messages. Determining the interval delay may include dynamically determining the delay.

In general, in another aspect, the disclosure describes a method of handling guaranteed messages received at a message-orient middleware server over a network. The method includes storing, in a log queue in non-persistent storage, guaranteed messages received from at least one client as the guaranteed messages are received. The method also includes removing guaranteed messages from the non-persistent storage as the guaranteed messages are delivered, dynamically determining a delay time period, and storing guaranteed messages stored in the non-persistent storage in persistent storage after the determined delay period. The method also includes transmitting a guarantee acknowledgement message to a client that sent a received message indicating that the message will not be lost by the server.

Embodiments may include one or more of the following features. Transmitting the guarantee acknowledgement message may include transmitting the guarantee acknowledgement message for a delivered guaranteed message if the guaranteed message is not persistently stored, the guarantee acknowledgement message otherwise being transmitted when the message is persistently stored.

In general, in another aspect, the disclosure describes a computer program product, disposed on a computer readable medium, for handling messages received at a server. The computer program includes instructions for causing a server processor to store, in a non-persistent storage, messages received from at least one client as the messages are received, remove messages from the non-persistent storage as the messages

are delivered, and save messages stored in the non-persistent storage to persistent storage after a delay period.

In general, in another aspect, the disclosure describes a message oriented middleware server. The server includes non-persistent storage, persistent storage, at least one processor, and instructions. The instructions cause the server processor to store, in the non-persistent storage, messages received from at least one client as the messages are received, remove messages from the non-persistent storage as the messages are delivered, and save messages stored in the non-persistent storage to persistent storage after a delay interval.

Advantages of the techniques described herein will become apparent in view of the following description, including the figures.

Brief Description of the Drawings

FIGs. 1 to 6 are diagrams illustrating operation of a messaging system.

FIGs. 7 to 9 are flow-charts illustrating operation of a messaging system.

Detailed Description

FIGs. 1 to 6 illustrate operation of a messaging system 100 that offers high performance and can reduce the resources needed to handle messaging between different applications 108-114. As shown, the server 100 includes both persistent storage 106 (i.e., non-volatile storage) and non-persistent storage 102 (i.e., volatile storage). The different types of storage 102, 106 offer different storage features. For example, while data stored in persistent storage 106 typically survives system 100 crashes, non-persistent storage 104 usually offers much faster access to data. Or, more concretely, storing and retrieving

data to and from a hard disk, an example of persistent storage 106, usually takes much longer than storing and retrieving data from R.A.M. (Random Access Memory), an example of non-persistent storage 102.

To improve message handling, for example, of guaranteed messages, the system 100 shown can balance the performance cost of storing messages in persistent storage 106 with the protection offered by doing so. For example, in the case of guaranteed messages, the messaging server 100 can add the messages to a queue 104, known as a log queue, in non-persistent storage 102. The system 100 continually removes messages from the log queue 104 as the server 100 delivers the messages. After a delay interval, the system 100 begins saving undelivered guaranteed messages to persistent storage 106. Since the system 100 will often deliver many guaranteed messages before expiration of the delay interval, the system 100 can avoid saving a large number guaranteed message to persistent storage 106. Storing fewer messages in persistent storage 106 can increase the message handling speed of the system 100 and can reduce the amount of persistent storage 106 needed. For guaranteed messages that remain undelivered beyond an expiration of a delay interval, however, the system 100 still provides the reliability of persistent storage.

To guarantee a message, the system 100 transmits a guarantee acknowledgement message to a message sender after message delivery or persistent storage of the message. Until the system 100 transmits a guarantee acknowledgement message, the system 100 has not guaranteed that the system 100 will not lose the sender's message. Thus, in the event the system 100 crashes before delivery or persistent storage of a message, the server 100 will not transmit an acknowledgement message guaranteeing the sender's message.

In greater detail, FIG. 1 shows a messaging system 100 and four different client applications 108-110. The messaging system 100 and clients 108-110 may communicate over a network such as the Internet.

As shown, two different clients, application A 108 and application B 110, send messages 116, 118 to the messaging system server 100. The messages 116, 118 may be designated as guaranteed messages, for example, by information included in the messages 116, 118. Though client applications 108, 110 may refrain from sending additional messages until receiving an acknowledgement of a guaranteed message from the server 100, the server 100 awaits either delivery or persistent storage of the message before transmitting the guarantee acknowledgement.

As shown in FIG. 2, the server 100 adds the received messages 116, 118 to the log queue 104 in non-persistent storage 102. Such adding may feature a copy of the message. Alternatively, adding the message to the log queue 104 may feature a reference or pointer to a memory location storing the message.

The server 100 may feature other queues (not shown). For example, in JMS (Java Messaging Service), a standard design for message-oriented middleware, senders and receivers communicate via queues. For instance, a sending application may send a message specifying a queue. A receiving application requests a message from the same queue. The messaging system 100 may handle communication by such queuing or use a wide variety of other communication techniques such as topic-based message handling using publish/subscribe, and so forth. Regardless of the technique, however, the server 100 can maintain the log queue 104 described herein.

As shown in FIG. 3, application C 112 transmits a request 120 for delivery of message #1 116. Again, how the application

112 makes such a request depends on the communication techniques supported by the server 100. As shown in FIG. 4, the server 100 transmits message #1 116 to application C 112.

As shown in FIG. 5, after receipt of message #1 116, application C 112 transmits a receipt acknowledgement message 122 back to the server 100. The system 100, in turn, transmits a guarantee acknowledgement message 124 to the sending client, application A 124, since delivery of the message 116 enables the system 100 to guarantee that it will not lose the message 116 prior to delivery. The system 100 may also transmit a message (not shown) indicating confirmed receipt of a message by a receiving application. As shown, the system 100 removes message #1 116 from the log queue 104 by deleting the message, message reference, or otherwise indicating that the server 100 need not save the message 116 to persistent storage 106.

As shown in FIG. 6, after a delay interval, the server 100 saves guaranteed messages, such as message #2 118, remaining in the log queue 104 to persistent storage 106. After saving the message 118 to persistent storage, the server 100 can remove the message 118 from the log queue 104. As shown, the server 100 can then transmit a guarantee acknowledgment message 126 to the client application 110 that sent the message 118 that has since been saved to persistent storage 106. Again, saving the message 118 to persistent storage 106 enables the server 100 to guarantee that the server 100 will not lose the message 118. After saving message 118, and, potentially, other guaranteed messages in the log-queue 104, the server 100 can begin waiting another delay interval before, again, saving messages to persistent storage 106. Since the server 100 may deliver many guaranteed messages during the delay, the delay can reduce messages written to the persistent storage 106. This reduced file activity can free the server 100 to devote resources to

other tasks such as message delivery and obtain a higher rate of message delivery overall.

The server 100 can retrieve persistently stored messages in the event of a problem such as a system 100 crash. For example, the server 100 can retrieve the persistently stored messages and replace them on an appropriate JMS queue. However, when the server 100 finally delivers messages retrieved from persistent storage 106, the server 100 may not send another guarantee acknowledgement message since the server 100 did so when storing the message in persistent storage 106.

Though FIGs. 1 to 6 illustrate server 100 operation as a series of stages, many of the operations shown can occur concurrently. For example, the server 100 may continually receive new messages for delivery and add these messages to the log-queue 104 as the server 100 simultaneously sends messages on to receiving applications. As this receiving and sending continues, the server 100 can intermittently save messages in the log queue 104 to persistent storage 106. Even as the server 100 saves messages to persistent storage 106, the server 100 may continue to add newly received messages to log queue 104 and remove delivered messages from the log queue 104. To ensure that messages wait in the log queue 104 for at some interval before being persistently stored, messages added to the log queue 104 after the server 100 begins saving messages to persistent storage may not be saved to persistent storage 106 until expiration of the next delay interval.

FIGs. 7 to 9 illustrate a sample implementation that features different processing threads provided by the messaging server 100. Again, the server 100 can execute these threads concurrently. It should be noted that the system may be implemented in a wide variety of ways. Other implementations

need not use threads. For example, a "monolithic" procedure may handle the tasks shown as being performed by different threads.

In greater detail, FIG. 7 illustrates a thread 200 that processes messages received 202 from client applications. The thread 200 adds 204 the received messages to the log queue 104 described above.

FIG. 8 illustrates a log flush thread 220 that saves messages in the log-queue to persistent storage 106 after a delay interval. The server 100 may use the same, constant delay interval time and again. For example, the server 100 can determine a delay interval by accessing a memory location storing a user specified delay. Alternatively, as shown, the thread 220 can dynamically determine 222 a delay interval, for example, based on messages handled by the server 100.

In practice, a variable delay can enhance overall message throughput as the messaging environment changes. For example, a variable delay can improve performance when the number of messaging clients changes over time. That is, at low client counts, a persistent storage sub-system 106 may be fast enough to satisfy message requests. Thus, in such environments, immediately storing messages in persistent storage 106 and sending a guarantee acknowledgment message enables a sending client awaiting acknowledgment to proceed sooner than a deferred delivery of the acknowledgment message might allow. Thus, where few clients are sending and/or receiving messages, a small delay may improve overall system performance.

When a system 100 handles messages for a large number of senders and receivers, reducing the overhead of accessing persistent storage for the increased number of messages 106 can improve system 100 performance. Thus, a longer delay can increase message delivery speed by allowing more time for

delivery of messages before saving them to persistent storage 106.

An appropriate amount of delay, however, may also depend on the current messaging environment (e.g., number of clients, average message size) of the server 100 and attributes of the storage system (e.g., the speed of access to persistent 106 and non-persistent storage 102, and the amount of persistent 106 and non-persistent storage 102 available).

In some embodiments, a server 100 determines 222 a variable delay based on one or more metrics, such as metrics based on message traffic handled by the server 100. For example, a metric may correspond to the number of different clients sending and/or receiving messages. For instance, in the case of a metric based on the number of clients sending messages, a process can determine a sender identifier of messages received, for example, as received messages are added to the log queue 104. The server 100 adds the identifier to a list such as an array of configurable length or a linked list. Because the list may hold a limited number of entries, the oldest entry may be discarded as the new entry is added. The server 100 may use the list to determine the number of sending clients for a given set of messages in the log queue 104. Based on the number of different sending clients, the server 100 can select a delay interval varying between zero for a single client and a maximum value when the entire list contains unique entries. The upper and lower bounds of the delay value may be configured. Additionally, the delay value may vary linearly or non-linearly, relative to the client metric.

The server 100 can determine a wide variety of other metrics. For example, the server 100 may determine a metric based on the average message size for messages currently added to the log queue 104. For a metric indicating a larger average

message size, the server 100 may reduce the delay interval. Additionally, yet another metric may reflect the speed of persistent storage.

After, potentially, determining a delay interval 222, the thread 220 can suspend or otherwise wait 224 for the delay interval to expire. When the interval expires, the thread 220 can begin saving log-queued messages to persistent storage 106. In some embodiments, before the thread 220 begins saving messages to persistent storage 106, the thread 220 can identify the last position of the log queue. The thread 220 can be configured to not save messages after this position. Thus, although additional messages may be added to the log queue, the thread 220 will save those messages placed in the log-queue before the delay interval expired. Thus, messages received during the save operation may remain on the log-queue, if not delivered, until expiration of the next interval.

For messages in the queue requiring persistent storage (e.g., guaranteed messages) 226, the thread 220 can store 228 the messages in persistent storage and transmit 230 guarantee acknowledgement messages to the respective message senders. After handling the messages in the log-queue 104, the thread 220 (if so configured) can again determine a new delay interval and suspend operation as newly received messages accumulate in the log queue 104 before resumption of the thread 220 after the determined interval.

FIG. 9 illustrates a process 250 for delivering messages. As shown, the process 250 receives a message request 252 from a client application. Such a request may include different information depending on the type of communication (e.g., queue-based, topic-based for publish/subscribe, and so forth). The process 250 can then deliver the message 262. Depending on the system 100, the system 100 may deem the message delivered based

on different criteria. For example, the system 100 may not consider a message delivered until receipt of an acknowledgement message from the recipient. Alternatively, the process 250 may deem a message delivered when transmitted, or even when the request for the message arrives. After message delivery 262, the illustrated process 250 removes 264 (e.g., delete or mark as delivered) the delivered message from the queue 104 and transmits 266 a "guarantee" acknowledgment message to the message's original sender.

The techniques described herein are not limited to any particular hardware or software configuration; they may find applicability in any computing or processing environment. The techniques may be implemented in hardware or software, or a combination of the two. Preferably, the techniques are implemented in computer programs executing on programmable computers that each include a processor, a storage medium readable by the processor (including volatile and non-volatile memory and/or storage elements), at least one input device, and one or more output devices.

The program(s) may be implemented in high level procedural or object oriented programming language to communicate with a computer system. However, the programs can be implemented in assembly or machine language, if desired. In any case the language may be compiled or interpreted language.

The computer program(s) may be stored on a storage medium or device (e.g., CD-ROM, hard disk, or magnetic disk) that is readable by a general or special purpose programmable computer for configuring and operating the computer when the storage medium or device is read by the computer to perform the procedures described herein. The system may also be considered to be implemented as a computer-readable storage medium, configured with a computer program, where the storage medium so

configured causes a computer to operate in a specific and predefined manner.

Other embodiments are within the scope of the following claims.